Announcements

• Lab 2 is due on Monday Sept 21st

• Lab 3 is posted
  – Due September 30th

Lab 3
Today’s Topics

- MVC - from last class
- Views
- Drawing
- Text & Images
- Animation

Model, View, Controller
Model, View, Controller

Model

- Manages the app data and state
- Not concerned with UI or presentation
- Often persists somewhere
- Same model should be reusable, unchanged in different interfaces
**View**

- Present the Model to the user in an appropriate interface
- Allows user to manipulate data
- Does not store any data
  - (except to cache state)
- Easily reusable & configurable to display different data

**Controller**

- Intermediary between Model & View
- Updates the view when the model changes
- Updates the model when the user manipulates the view
- Typically where the app logic lives
Model, View, Controller

Model, View, Controller

Model Object

Controller
outlets
actions

View
Views

View Fundamentals

• Rectangular area on screen
• Draws content
• Handles events
• Subclass of UIResponder (event handling class)
• Views arranged hierarchically
  – every view has one superview
  – every view has zero or more subviews
View Hierarchy - UIWindow

- Views live inside of a window
- UIWindow is actually just a view
  - adds some additional functionality specific to top level view
- One UIWindow for an iPhone app
  - Contains the entire view hierarchy
  - Set up by default in Xcode template project

View Hierarchy - Manipulation

- Add/remove views in IB or using UIView methods
  - (void)addSubview:(UIView *)view;
  - (void)removeFromSuperview;
- Manipulate the view hierarchy manually:
  - (void)insertSubview:(UIView *)view atIndex:(int)index;
  - (void)insertSubview:(UIView *)view belowSubview:(UIView *)view;
  - (void)insertSubview:(UIView *)view aboveSubview:(UIView *)view;
  - (void)exchangeSubviewAtIndex:(int)index withSubviewAtIndex:(int)otherIndex;
View Hierarchy - Ownership

- Superviews retain their subviews

- Not uncommon for views to only be retained by superview
  - Be careful when removing!
  - Retain subview before removing if you want to reuse it

- Views can be temporarily hidden
  theView.hidden = YES;

View-related Structures

- CGPoint
  - location in space: \(\{ x, y \}\)
  - sometimes used as an origin

- CGSize
  - dimensions: \(\{ \text{width}, \text{height} \}\)

- CGRect
  - location and dimension: \(\{ \text{origin}, \text{size} \}\)
### Rects, Points and Sizes

![Diagram showing Rects, Points and Sizes](image)

#### View-related Structure

<table>
<thead>
<tr>
<th>Creation Function</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGPointMake(x,y)</td>
<td>CGPoint point = CGPointMake(100.0, 200.0);</td>
</tr>
<tr>
<td></td>
<td>point.x = 300.0;</td>
</tr>
<tr>
<td></td>
<td>point.y = 30.0;</td>
</tr>
<tr>
<td>CGSizeMake(width, height)</td>
<td>CGSize size = CGSizeMake(42.0, 11.0);</td>
</tr>
<tr>
<td></td>
<td>size.width = 100.0;</td>
</tr>
<tr>
<td></td>
<td>size.height = 72.0;</td>
</tr>
<tr>
<td>CGRectMake(x,y,width, height)</td>
<td>CGRect rect = CGRectMake(100.0, 200.0, 42.0, 11.0);</td>
</tr>
<tr>
<td></td>
<td>Rect.origin.x = 0.0;</td>
</tr>
<tr>
<td></td>
<td>Rect.size.width = 50.0</td>
</tr>
</tbody>
</table>
UIView Coordinate System

- Origin in upper left corner
- y axis grows downwards

Location and Size

- View’s location and size expressed in two ways
  - Frame is in superview’s coordinate system
  - Bounds is in local coordinate system

- View A frame:
  - Origin: 0,0
  - Size: 550 x 400

- View A bounds:
  - Origin: 0,0
  - Size: 550 x 400

- View B frame:
  - Origin: 200, 100
  - Size: 200 x 250

- View B bounds:
  - Origin: 0,0
  - Size: 200 x 250
Frame and Bounds

- Which to use?
  - Usually depends on the context

- If you are using a view, typically you use frame

- If you are implementing a view, typically you use bounds

- Matter of perspective
  - From outside it’s usually the frame
  - From inside it’s usually the bounds

- Examples:
  - Creating a view, positioning a view in superview - use frame
  - Handling events, drawing a view - use bounds

Creating Views
Where do views come from?

- Commonly Storyboard or Interface Builder
- Drag out any of the existing view objects (buttons, labels, etc)
- Or drag generic UIView and set custom class

Manual Creation

- Views are initialized using -initWithFrame:
  - CGRect frame = CGRectMake(0, 0, 200, 150);
  - UIView *myView = [[UIView alloc] initWithFrame:frame];

- Example:
  CGRect frame = CGRectMake(20, 45, 140, 21);
  UILabel *label = [[UILabel alloc] initWithFrame:frame];

  [window addSubview:label]; // retain count for label increased by 1
  [label setText:@"Number of sides:"];
  [label release]; // label now retained by window
Defining Custom Views

- Subclass UIView

- For custom drawing, you override:
  - (void)drawRect:(CGRect)rect;

- For event handling, you override:
  - (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event;
  - (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event;
  - (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event;

Drawing Views
-(void)drawRect:(CGRect)rect

- [UIView drawRect:] does nothing by default
  - If not overridden, then backgroundColor is used to fill

- Override - drawRect: to draw a custom view
  - rect argument is area to draw

- When is it OK to call drawRect:?

---

Be Lazy

- drawRect: is invoked automatically
  - Don’t call it directly!

- Being lazy is good for performance

- When a view needs to be redrawn, use:
  - (void)setNeedsDisplay;

- For example, in your controller:
  - (void)setNumberOfSides:(int)sides {  
    numberOfSides = sides;  
    [polygonView setNeedsDisplay];  
  }
CoreGraphics and Quartz 2D

- UIKit offers very basic drawing functionality
  - UIRectFill(CGRect rect);
  - UIRectFrame(CGRect rect);

- CoreGraphics: Drawing APIs

- CG is a C-based API, not Objective-C

- CG and Quartz 2D drawing engine define simple but powerful graphics primitives
  - Graphics context
  - Transformations
  - Paths
  - Colors
  - Fonts
  - Painting operations
Graphics Contexts

- All drawing is done into an opaque graphics context
- Draws to screen, bitmap buffer, printer, PDF, etc.
- Graphics context setup automatically before invoking `drawRect`:
  - Defines current path, line width, transform, etc.
  - Access the graphics context within `drawRect` by calling
    `(CGContextRef) UIGraphicsGetCurrentContext(void);
  - Use CG calls to change settings
- Context only valid for current call to `drawRect`:
  - Do not cache a CGContext!

CG Wrappers

- Some CG functionality wrapped by UIKit
  - UIColor
    - Convenience for common colors
    - Easily set the fill and/or stroke colors when drawing
      ```
      UIColor *redColor = [UIColor redColor];
      [redColor set];
      // drawing will be done in red
      ```
  - UIFont
    - Access system font
    - Get font by name
      ```
      UIFont *font = [UIFont systemFontOfSize:14.0];
      [myLabel setFont:font];
      ```
Simple drawRect: example

- Draw a solid color and shape

-(void)drawRect:(CGRect)rect {
CGRect bounds = [self bounds];

[[UIColor grayColor] set];
UIRectFill (bounds);

CGRect myShape = CGRectMake (10, 10, 50, 100);
[[UIColor redColor] set];
UIRectFill (myShape);

[[UIColor blackColor] set];
UIRectFrame (myShape);
}

What shape is this?

Drawing More Complex Shapes

- Common steps for drawRect:
  - Get current graphics context
  - Define a path
  - Set a color
  - Stroke or fill path
  - Repeat, if necessary
Paths

- CoreGraphics paths define shapes

- Made up of lines, arcs, curves and rectangles

- Creation and drawing of paths are two distinct operations
  - Define path first, then draw it

CGPath

- Two parallel sets of functions for using paths
  - CGContext "convenience" throwaway functions
  - CGPath functions for creating reusable paths

<table>
<thead>
<tr>
<th>CGContext</th>
<th>CGPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGContextMoveToPoint</td>
<td>CGPathMoveToPoint</td>
</tr>
<tr>
<td>CGContextLineToPoint</td>
<td>CGPathAddLineToPoint</td>
</tr>
<tr>
<td>CGContextAddArcToPoint</td>
<td>CGPathAddArcToPoint</td>
</tr>
<tr>
<td>CGContextClosePath</td>
<td>CGPathCloseSubPath</td>
</tr>
</tbody>
</table>

And so on....
Simple Example

```c
-(void)drawRect:(CGRect)rect {
    CGContextRef context = UIGraphicsGetCurrentContext();
    [[UIColor grayColor] set];
    UIRectFill ([self bounds]);
    CGContextBeginPath (context);
    CGContextMoveToPoint (context, 75, 10);
    CGContextAddLineToPoint (context, 10, 150);
    CGContextAddLineToPoint (context, 160, 150);
    CGContextClosePath (context);
    [[UIColor redColor] setFill];
    [[UIColor blackColor] setStroke];
    CGContextDrawPath (context, kCGPathFillStroke);
}
```

Demo - HelloPoly
More Drawing Information

- UIView Class Reference
- CGContext Reference
- “Quartz 2D Programming Guide”
- Lots of samples in the iPhone Dev Center

Images & Text
**UIImage**

- UIKit class representing an image

**Creating UIImages:**
- Fetching image in application bundle
  - Use `+[UIImage imageNamed:(NSString *)name]`
  - Include file extension in file name, e.g. `@"myImage.jpg"`
- Read from file on disk
  - Use `-[UIImage initWithContentsOfFile:(NSString *)path]`
- From data in memory
  - Use `-[UIImage initWithData:(NSData *)data]`

**Text, Images, and UIKit views**
Constructing Views

- How do I implement this?

- Goal
  - PolygonView that displays shape as well as name

- Initial thought
  - Have PolygonView draw the text
  - Inefficient when animating

- Instead use UILabel!
  - Tastes great
  - Less filling

UILabel

- UIView subclass that knows how to draw text

- Properties include:
  - font
  - textColor
  - shadow (offset & color)
  - textAlignment
**UIImage**

- **UIView that draws UIImages**

- **Properties include:**
  - image
  - animatedImages
  - animatedDuration
  - animatedRepeatCount

- **contentMode property to align and scale image wrt bounds**

---

**View Properties & Animation**
Animating Views

- What if you want to change layout dynamically?
- For example, a switch to disclose additional views...

![Image of pentagon with animation controls]

UIView Animations

- UIView supports a number of animatable properties
  - frame, bounds, center, alpha, transform

- Create “blocks” around changes to animatable properties

- Animations run asynchronously and automatically
Other Animation Options

• Additional animation options
  – delay before starting
  – start at specific time
  – curve (ease in/out, ease in, ease out, linear)
  – repeat count
  – autoreverses (e.g. ping pong back and forth)

View Animation Example

-(void)showAdvancedOptions {
  // assume polygonView and optionsView
  [UIView beginAnimations:@"advancedAnimations" context:nil];
  [UIView setAnimationDuration:0.3];

  // make optionsView visible (alpha is currently 0.0)
  optionsView.alpha = 1.0;

  // move the polygonView down
  CGRect polygonFrame = polygonView.frame;
  polygonFrame.origin.y += 200;
  polygonView.frame = polygonFrame;

  [UIView commitAnimations];}
Knowing When Animations Finish

- **UIView animations allow for a delegate**
  ```
  [UIView setAnimationDelegate:myController];
  ```

- **myController will have callbacks invoked before and after**
  ```
  -(void)animationWillStart:(NSString *)animationID
                context:(void *)context;
  ```
  ```
  -(void)animationDidStop:(NSString *)animationID finished:(NSNumber *)finished
             context:(void *)context;
  ```

- **Can provide custom selectors if desired, for example**
  ```
  [UIView setAnimationWillStartSelector:@selector(animationWillStart)];
  [UIView setAnimationDidStopSelector:@selector(animationDidStop)];
  ```

How does it work?

- **Utilizes Core Animation**

- **Hardware accelerated rendering engine**

- **UIViews are backed by “layers”**

- **-drawRect: results are cached**
  - Cached results used to render view
  - `-drawRect: called only when contents change`
  - Layers maintained in separate hierarchy managed by separate process

- **Property animations done automatically by manipulating layers**
View Transforms

- Every view has a transform property
  - used to apply scaling, rotation and translation to a view
- Default “Identity transform”

- CGAffineTransform structure used to represent transform

- Use CG functions to create, modify transforms

```cpp
CGAffineTransformFunctions(small example set...)
CGAffineTransformScale(transform, xScale, yScale)
CGAffineTransformRotate(transform, angle)
CGAffineTransformTranslate(transform, xDelta, yDelta)
```

More Animation Information

- iPhone OS Programming Guide
  - “Modifying Views at Runtime” section
  - Core Animation Programming Guide
More Examples

Demo – Animations

https://github.com/bobmccune/Core-Animation-Demos

http://code4app.net/ios/ShareOfCoreAnimation/55112093e247417005932c4a
Final Project from Previous Semester